LAB 3

PROJECT EXPLORATION MISSION AND CYBOT-PC COMMUNICATIONS (MISSION 1)

INTRODUCTION

In this lab, you will conduct your first mission with the CyBot mobile robot. You have been training with the CyBot platform, and it's time to use what you know to accomplish a mission in which the CyBot moves through a test field, detects and avoids objects, and drives to the smallest object. These represent the minimum capabilities you will implement in your final lab project.

In Lab 2, you wrote code to move the robot. In this lab, you will first write code to send messages between the CyBot and the PC. You will use functions from a precompiled library to send and receive bytes – the CyBot UART library. A precompiled library has a header (.h) file and a library (.lib) file. The header file has function prototypes and other code that you as the programmer can view. The library file is a binary file that has executable code that you cannot view.

You will also use code from a precompiled library to scan for objects in front of the CyBot – the CyBot Scan library. The code in this library scans the area in front of the CyBot, rotating the sonar (ultrasonic, PING) and infrared (IR) sensors 180 degrees using the servo motor. In later labs, you will learn how to write your own code that interfaces with the sensors and motor using microcontroller peripherals. For the project, you will use your own code – based on code that you will be writing in later labs. Until then, the Scan library will let you start experimenting with more capabilities of the CyBot in the test field.

In this lab, you will develop an embedded application that can: 1) identify the smallest width (tall) object in the test field, and then 2) navigate to that object while avoiding obstacles (optional for now). To help guide you through this mission, we have divided the mission into three parts.

REFERENCE FILES

The following reference files will be used in this lab:

- cyBot_Scan.h, header file for precompiled library to scan for objects
- libcybotScan.lib: precompiled library for scanning using sensors (**note: must change extension of file from .txt to .lib after copying**, do not open the file, download/copy only)
- cyBot_uart.h, header file for precompiled library to communicate between the CyBot and PC
- libcybotUART.lib: precompiled library for CyBot-PC UART communication (**note: must change extension of file from .txt to .lib after copying**, do not open the file, download/copy only)

- lcd.c, program file containing various LCD functions
- lcd.h, header file for lcd.c
- timer.c, program file containing various wait commands
- timer.h, header file for timer.c
- open_interface.c, API functions for basic Open Interface functions
- open_interface.h, header file for open_interface.c
- Cybot baseboard and LCD schematics: Cybot-Baseboard-LCD-Schematic.pdf
- **PuTTY UART and WiFi Board Quick Reference Sheet.pdf**
- **Servo Calibration Reference Sheet.pdf**

The code files are available to download.

PRELAB

See the prelab assignment in Canvas and submit it prior to the start of lab.

STRUCTURED PAIRING

You are expected to continue to use structured pairing in this lab and in future labs. It was introduced in Lab 2.

MISSION GROUND RULES

1. Allowed sensors
   a. `cyBot_Scan()`, which uses the ultrasonic (PING) and infrared (IR) sensors on the CyBot
   b. iRobot sensors accessed through Open Interface, including
      i. distance
      ii. angle
      iii. bumpLeft, bumpRight
      iv. cliffFrontLeft, cliffFrontRight

2. Objects in the test field
   a. Short objects: detected by the bump sensors
   b. Holes: detected by the cliff sensors
   c. Tall objects: detected by `cyBot_Scan()`. The CyBot should never collide with a tall object. It should get no closer than 2 cm from a tall object.

See Figure 1 below for illustrations of possible objects in a test field. Note that you will use a simple test field for this lab. You will position objects in front of the CyBot (within the 180-degree scan area). You should place tall cylinder objects at equal distances from the CyBot and with enough separation between them (except for a composite object). You will understand the reasons for these simplifications as you work on this lab. The test field for the project will not be simplified in this way. You can start experimenting with more complicated test fields after your system works with simpler ones.

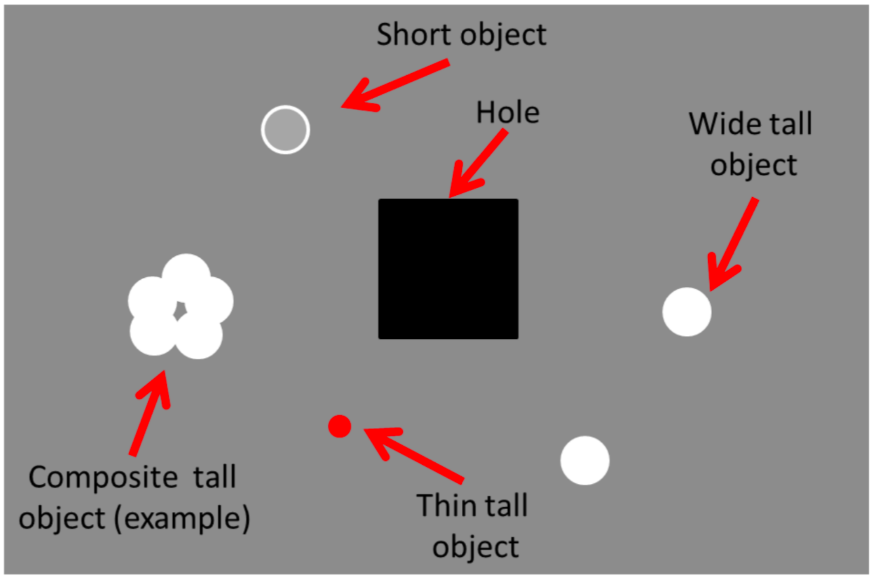


Figure 1: Example test field

An actual test field used with a CyBot in the lab project in past semesters is shown in the videos below. You may have watched these earlier in the semester. These links are also in Canvas at the bottom of the Lab Resources page.

- https://www.youtube.com/watch?v=ulidN0rs1NA (early generation CyBot)
- https://www.youtube.com/watch?v=OU5m58bhZ6Y (later generation CyBot)

Notice the sensors scanning the region in front of the robot.

Reminder: Copy your CCS project from a previous lab and modify the main program for this lab. As before, the main program will use a while loop that continuously calls functions to implement initialization, control, status and data processing tasks in software. Copy/download the new library files (header file and lib file). Write new functions as needed to implement functionality for this lab.

PART 1: CYBOT COMMUNICATION USING PUTTY

In this part of the lab, you will write your first application to send messages to the CyBot from your desktop PC, and get messages back.

1. Look over cyBot_uart.h. This file describes three functions you will want to use to communicate between the PC and your program on the CyBot. The communication will use a UART serial port on the CyBot and a PC application called PuTTY. **Note: pay special attention to the descriptions of these three functions and the implications of their behavior.** In particular, the cyBot_getByte() function is a "blocking" function that

will not return a value until a byte is received from PuTTY. Inside the function implementation (not visible to you as a precompiled library), there is a busy-wait loop that is checking for a byte being received. Thus, a user needs to press a key in PuTTY, and then the ASCII character value of the key pressed is returned by the function.

2. Read the quick reference sheet for configuring PuTTY on the PC. For this part of the lab, you can try the UART serial connection and/or the WiFi network connection. For later parts, if the CyBot is placed in the test field, you must use WiFi.

3. Press a key on the PC using the PuTTY application to send a character from the PC to the CyBot (received by the CyBot), and then display the character on the CyBot's LCD screen.

4. Send a message back to PuTTY from the CyBot after receiving a character. For example, when you press the 'm' key in PuTTY, and 'm' is received by the CyBot, send a message to the PC, such as "*Got an m*". Hint: Write your own function to send a string to PuTTY. Your function should call the cyBot_sendByte() function (given in the cyBot_uart.h header file) to send the characters in the string one by one. Before calling your send string function, if needed, you can use the C standard library function sprintf() to format variables into a string. This approach will be very useful in later parts of the lab to send data from your CyBot to PuTTY.

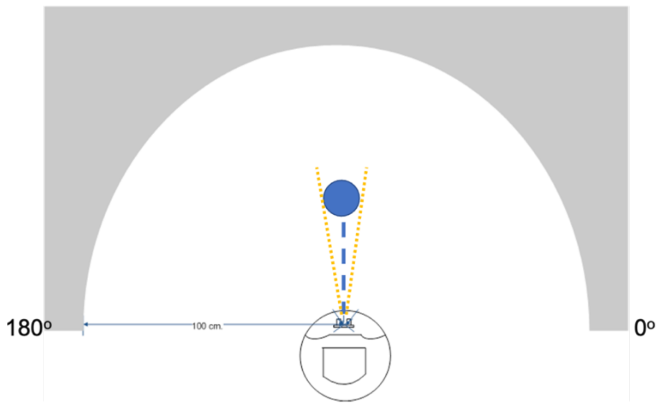
CHECKPOINT:

You should be able to send and receive messages between the CyBot and PC.

PART 2: COLLECT SENSOR DATA AND IDENTIFY OBJECTS IN THE DATA

For this part of the lab, place one tall object in front the CyBot. Looking forward from the front of the CyBot, the 0-degree position is to the right, and 180 is to the left.

For example, in the figure below, one object is shown in the 90-degree position. The blue dashed line is the distance to the object. The yellow dotted lines correspond to the positions (in degrees) of the edges of the object. When a sensor is scanned across a range of degrees, the sensor data will change and become more uniform between the edges of an object, as "seen" by the sensor. The distance to an object can be determined midway between the edges of an object. The edges can also be used to estimate or calculate the width of an object.



Review the functions and descriptions given in cyBot_Scan.h. This file lists several functions you will use to initialize the devices used in the scan operation, position the PING and IR sensors, get data from the sensors, and **as needed, calibrate the position of the servo motor**. The PING sensor data variable is a distance measurement in cm, whereas the IR sensor data is not a distance value. The IR sensor data is a digital value for the sensor's output voltage, which is inversely related to the distance, from which distance would be calculated using the sensor characteristics (you'll learn more about this later).

IMPORTANT: For the purposes of this lab, you only need to use the PING sensor data returned from the scan function (you do not need to use the IR data). As shown in the PING datasheet, the PING sensor provides accurate distance measurements for objects in the range of 2 cm to 3 meters. The actual range may be less due to other factors and can be tested in the field. Objects outside of this range and sensor readings outside of this range are not valid.

The PING sensor is useful to quickly detect an object and get an accurate distance measurement. However, the sound wave of the PING sensor detects an object over a wider range than the object itself, thus the positions of the edges identified with PING data may not be very accurate. Nonetheless, PING data may be sufficient to determine whether one object's width is smaller or larger than another. (In future labs, you'll see that the IR sensor, with its narrower beam of light, is useful to get accurate positions of the edges of an object. IR sensor data will be used to calculate an accurate actual width of an object.)

Here is a sample display of PING sensor data from a partial scan (data depend on your test field):

```
Degrees    PING Distance (cm)
10         140.75
12         161.73
14         161.16
16         182.51
18         121.37
20         156.91
22         134.15
24         160.52
26         28.84
28         27.52
```

```
30          27.06
32          26.76
34          26.76
36          26.51
38          25.66
40          25.60
42          25.99
44          25.62
46          25.59
48          25.90
50          25.62
```

Looking at the sample data above, it appears there could be an object that is about 26 cm in front of the CyBot.

PART A: Use Calibration to Accurately Position the Servo

1. Use cyBOT_Scan() to move the servo motor to 90 degrees.
2. If the servo did not move to the position specified in your code, several factors may cause this. Thus, to accurately position the servo, calibration may be needed. In this part of the lab, you will manually perform a calibration process to determine calibration values to use in your program.
   a. Browse the section of cyBot_Scan.h that describes the calibration code. Note that you will call the calibration function in your program, which will return calibration values that you will then assign to variables in your program. The calibration variables are then used by other scan library code to accurately position the servo.
   b. **Read the servo calibration reference sheet**. **Follow the instructions** to find values to assign to the calibration variables in your main program.
   c. Repeat step #1 above and confirm that the actual position of the servo more closely matches the specified position. Try different positions. In other words, you are verifying that your servo calibration is helping your program more accurately position the servo.

PART B: Scan to Collect Sensor Data and Send to the PC

1. Start with a single tall object in front of the CyBot (e.g., directly in front at the 90-degree position). Use cyBOT_Scan() and write code to scan a range of degrees, such as 45 to 135 degrees, collecting sensor data every 2-5 degrees. Start the scan when an ‘m’ is sent from PuTTY to the CyBot. Display the angle and sensor information returned by the scan in PuTTY. See the sample display of information in PuTTY above.
2. Change the placement of the object in the test field and scan again. Confirm that your data show a new position (range of degrees) where the object is being seen by the sensor.
3. Confirm that your data show the correct distance to the object.

Tip: In order to format your data in PuTTY, use escape sequences in a formatted string, such as ‘\r’ to return, ‘\t’ for tab, and ‘\n’ for newline.

Tip: To save data from PuTTY into a file, you can set up "Session > Logging," and under filename, click "Browse" to specify where you want the output saved. Data displayed in PuTTY will then also be written to the file.

CHECKPOINT:

The object information is displayed in PuTTY.

PART 3: DETECT THE SMALLEST WIDTH OBJECT

For this part, the test field should have 2 or more tall objects that are a) spaced apart (so that the CyBot PING sensor "sees" them as different objects) and b) positioned about the same distance from the CyBot. One object should have a smaller width than the others. Additionally, some of the tall objects could be grouped together to create a wider object, as shown in Figure 1 (i.e., composite object). All objects should be located in front of the CyBot so that a 180-degree scan sees all objects.

1. Based on a completed scan, determine the following for each objected detected and display to PuTTY:
    a. Object number
    b. Angle at which an object is detected (e.g., position of the middle of an object in degrees)
    c. Distance to an object in cm
    d. Radial (or angular) width of an object in degrees (i.e., the number of degrees within the scan in which the object appears; e.g., if the scan starts seeing an object at angle 30 degrees and stops seeing it at 35 degrees, its radial width is 5 degrees)
2. Confirm that the data collected and displayed matches the state of the test field.
3. Position the CyBot sensor to point in the direction of the object with the smallest radial width.

Note that radial width is different than actual linear width. Geometry or trigonometry can be used to calculate or estimate the linear width (e.g., see arc length of a circle). It's okay to use radial width in this lab. In the project, you will need to use linear width, because objects will be at varying distances from the CyBot in the test field.

As noted earlier, keep in mind that the PING sensor emits a sound wave, which is shaped like a cone, and thus PING sensor data make an object appear wider than it actually is. Nonetheless, PING data may be sufficient to determine whether one object's width is smaller or larger than another.

CHECKPOINT:

Object information is displayed in PuTTY, and the CyBot points to the object with the smallest width.

(Optional) Part 4: Drive to the Smallest Width Object (2 Bonus Points)

Drive the CyBot, based on what you type into PuTTY. For example: program the key ‘w’ to go forward, ‘x’ to go backward, ‘a’ to go left, and ‘d’ to go right. Using WiFi will allow you to drive the CyBot without being tethered to a cable.

Drive the CyBot toward the object with the smallest width. Drive to within 10 cm of the object.

You may write code that autonomously drives toward the smallest width object, or you may manually drive your CyBot using only the sensor data displayed in PuTTY (you are not allowed to watch the CyBot as you drive).

Extra bonus point (for a total of 3 bonus points): Add one or more short objects to the test field. These can only be detected by the bump sensors. Navigate around short objects that the CyBot encounters along the way.

Checkpoint:

Object information is displayed in PuTTY, and the CyBot moves to within 10 cm of the object with the smallest width.

Demonstrations:

1. **Functional demo of a lab milestone –** Specific milestone to demonstrate in Lab 3: Checkpoint for Part 3
2. **Debug demo using debugging tools to explain something about the internal workings of your system –** The TA will announce any specific debugging requirements at the start of lab; otherwise you will create your own debug demo based on your needs and interests in the lab.
3. **Q&A demo showing the ability to formulate and respond to questions –** This can be done in concert with the other demos.